Week 4
Internet of Things 1

Learn about Loops
Creating Functions

Pair up in threes.
-Yogi Berra

Contents

- Learning About Loops
  - How to perform repetitive tasks
  - How to use the for loop
  - How to use the while loop
  - How to use nested loops
- Creating Functions
  - Creating your own functions
  - Retrieving data from functions
  - Passing data to functions
  - Using lists with functions
  - Using functions in your Python Scripts

Learning About Loops

How to perform repetitive tasks

- A synonym for repetition is iteration. In the programming world, iteration is the process of performing a defined set of tasks repeatedly until either a desired result is achieved or the set of tasks has been performed a desired number of times.

`for` Loops

How to use the for loop

- The for loop construct is called a **count-controlled loop** because the loop's set of tasks will be performed a set number of times.
- The syntax structure of the for loop in Python is as follows:

```
for variable in (data_list):
    set_of_Python_statements
```

How to use the for loop

- A for loop
  - We have a data list to use [1, 2, 3, 4, 5]
  - The variable the_number will take one value at the time and will be applied in the statements of the loop.

Shell

```
>>>
>>> for the_number in [1, 2, 3, 4, 5]:
        print (the_number)


 1
 2
 3
 4
 5

>>>
```

How to use the for loop

- A for loop
  - You need to be careful about a couple of potential problems with the for loop structure.
    - Forget to put a colon at the end of your for loop
    - Not to use commas to separate your numeric data list.

Shell

```
>>>
>>> for the_number in [1, 2, 3, 4, 5]
    print (the_number)


   File "<pyshell>", line 1
     for the_number in [1, 2, 3, 4, 5]
                                      ^
 SyntaxError: invalid syntax
>>> for the_number in [12345]:
        print (the_number)


 12345
>>>
```

How to use the for loop

- A for loop
  - Python behaves as you would expect with data types in for loops.
  - Python also change the data type, as needed, in the assignment.

Shell

```
>>>
>>> for the_numer in [1, 5, 15, 9]:
        print (the_number)
        type (the_number)

 12345
<class 'int'>
 12345
<class 'int'>
 12345
<class 'int'>
 12345
<class 'int'>
>>> for the_number in [1, 5.5, 15, 'hello']:
        print (the_number)
        type (the_number)

 1
<class 'int'>
 5.5
<class 'float'>
 15
<class 'int'>
 hello
<class 'str'>
>>>
```

How to use the for loop

- A for loop
  - You can iterate using Strings in a List

Shell

```
>>>
>>> for the_word in ['Alpha', 'Charlie', 'Delta', 'Echo']:
        print (the_word)


Alpha
Charlie
Delta
Echo
```

How to use the for loop

- A for loop
  - You can iterate using variables in the Data List

```
Shell
>>>
>>> top_number = 10
>>> for the_number in [1, 2, 3, 4, top_number]:
        print (the_number)


 1
 2
 3
 4
 10

>>>
```

How to use the for loop

- A for loop
  - Instead of listing all the numbers individually in a data list, you can use the range function to create a contiguous number list for you.
  - The range function really shines when it's used in loops!
  - By the way…
    - The range function is not really a function. It is actually a data type that represents an immutable sequence of numbers.

```
Shell
>>>
>>> for the_number in range (5):
        print (the_number)

0
1
2
3
4

>>>
```

How to use the for loop

- A for loop
  - Using the range function causes a numeric data list to be created: [0, 1, 2, 3, 4]
  - The range function, by default, starts at 0 and the produces a number list all the way up to stop number minus 1
  - You can alter the behavior of the range function by including a start number.

Shell

```
>>>
>>> for the_number in range (1,5):
        print (the_number)


 1
 2
 3
 4

>>>
```

How to use the for loop

- A for loop
  - Variables can be used in place of the numbers in the range function.

Shell

```
>>> start_number = 3
>>> stop_number = 8
>>> for the_number in range (start_number, stop_number):
        print (the_number)


3
4
5
6
7

>>>
```

How to use the for loop

- A for loop
  - You can change the increment of the number list produced by the range function, you include a step number in your range arguments.
  - By default, the range function increments the numbers in the list by 1.

Shell

```
>>> for the_number in range (2, 9, 2):
        print (the_number)


  2
  4
  6
  8

>>>
```

`while` Loops

How to use the while loop

- The while loop construct is called a **condition-controlled loop** because the loop's set of tasks are performed until a desired condition is met.
- After the condition is met, the iterations stop.
- The syntax structure of the while loop in Python is as follows:

```
while condition_test_statement:
    set_of_Python_statements
```

How to use the while loop

- You can use a number or mathematical equation in a while loop's condition test statement.

Shell

```
>>> the_number = 1
>>> while the_number <= 5:
        print (the_number)
        the_number = the_number + 1


 1
 2
 3
 4
 5

>>>
```

How to use the while loop

- while loops are pretested, which means the test statement is run before the statement in the code block are executed.
- This is why the variable the_number has to have a value assigned to it before the while loop's test condition is executed.
- These types of loops are called pretested, or entry control, loops.

How to use the while loop

- Character string can be part of the while loop's condition test statement.

Shell

```
>>> list_of_names = ""
>>> the_name = "Start"
>>> while the_name != "":
        the_name = input ("Enter name: ")
        list_of_names = list_of_names + the_name


  Enter name: Paul
  Enter name: Anne
  Enter name:

>>>
```

How to use the while loop

- A nice tweak on the while loop is to include an optional else clause in the while loop.

Shell

```
>>>
>>> list_of_names = ""
>>> the_name = "Start"
>>> while the_name != "":
        the_name = input ("Enter name: ")
        list_of_names = list_of_names + the_name
    else:
        print(list_of_names)


  Enter name: Paul
  Enter name: Anne
  Enter name:
  PaulAnne

>>>
```

How to use the while loop

- An infinite loop, is a loop that “never ends”.
- This infinite loop can be created using a while loop.
- You need to add a break statement to this type of while loop to make it usable.

Shell

```
>>>
>>>
>>> list_of_names = ""
>>> the_name = "Start"
>>> while True:
        the_name = input ("Enter name: ")
        if the_name == "":
            break
        list_of_names += the_name
    else:
        print (list_of_names)

 Enter name: Paul
 Enter name: Anne
 Enter name:

>>>
```

How to use the while loop

- The else clause is not executed.
- This is because when you issue a break in a loop, any Python statement in the else clause are skipped.

Shell

```
>>> list_of_names = ""
>>> the_name = "Start"
>>> while True:
        the_name = input ("Enter name: ")
        if the_name == "":
            print (list_of_names)
            break
        list_of_names += the_name


  Enter name: Paul
  Enter name: Anne
  Enter name:
  PaulAnne

>>>
```

How to use nested loops

- A nested loop is a loop statement that is inside a loop statement.
- For example, a for loop used within the code block of a while loop would be a nested loop.

How to use nested loops

nested_loop.py

```python
# nested_loop.py - Demonstration of a nested loop.
##########################################################
#
# Find out how many club member names need to be entered
names_to_enter=int(input("How many Python club member names to enter? "))
#
# Loop to enter names:
for member_number in range (1, names_to_enter + 1):
    print()
    print("Member #" + str(member_number))
#
    first_name=""   # Intialize first_name
    middle_name=""  # Intialize middle_name
    last_name=""    # Intialize last_name
#
### Loop to get first name
    while first_name == "":
          first_name=input("First Name: ")
#
### Loop to get middle name
    while middle_name == "":
          middle_name = input("Middle Name: ")
#
### Loop to get last name
    while last_name == "":
          last_name = input("Last Name: ")
#
    # Display a member's full name
    print()
    print ("Member #", member_number, "is",
            first_name, middle_name, last_name)
```

Shell

```
>>>
>>> %cd /home/pi/Documents/scripts
>>> %Run nested_loop.py

  How many Python club member names to enter? 2

  Member #1
  First Name: Paul
  Middle Name: R
  Last Name: Tremblay

  Member # 1 is Paul R Tremblay

  Member #2
  First Name: Anne
  Middle Name: L
  Last Name: Rivard

  Member # 2 is Anne L Rivard

>>>
```

Creating Functions

Creating your own functions

- As you start writing more complex Python scripts, you'll find yourself reusing parts of code that perform specific tasks.
- Python provides a feature that lets you reuse the block of code.
- Any time you need to use that block of code in your script, you simple use the name you assigned to the function.
- This is referred to as calling the function.

Creating your own functions

- To create a function in Python, you use the def keyword followed by the name of the function, with parentheses:

```
def myfunction () :
    statement1
    statement2
    statement3
```

- Note the colon at the end of the statement.

Creating your own functions

functions.py *

```
def function():
    print ("This is an example of a function")

count = 1
while (count <= 5):
    function()
    count = count + 1

print ("This is the end of the loop.")
function()
print ("Now this is the end of the script")
```

Shell

```
>>> %Run functions.py

  This is an example of a function
  This is an example of a function
  This is an example of a function
  This is an example of a function
  This is an example of a function
  This is the end of the loop.
  This is an example of a function
  Now this is the end of the script

>>>
```

Creating your own functions

functions2.py

```python
count = 1
print ("This line comes before the function definition")

def function():
    print ("This is an example of a function")

while (count <= 5):
    function()
    count = count + 1

print ("This is the end of the loop.")
function2()
print ("Now this is the end of the script")

def function2():
    print ("This is an example of a misplaced function")

```

Shell

```
>>> %Run functions2.py

  This line comes before the function definition
  This is an example of a function
  This is an example of a function
  This is an example of a function
  This is an example of a function
  This is an example of a function
  This is the end of the loop.
  Traceback (most recent call last):
    File "/home/pi/Documents/scripts/functions2.py", line 12, in <module>
      function2()
  NameError: name 'function2' is not defined

>>>
```

Creating your own functions

- You need to be careful about functions names.
- Each function name must be unique.
- If you redefine a function, the new definition overrides the original function definition, without producing any error message.

functions3.py

```
def function():
    print ("This is the first definition of the function")

function()

def function():
    print ("This is a repeat definition of the function")

function()

print ("This is the end of the script")
```

Shell

```
>>> %Run functions3.py
  This is the first definition of the function
  This is a repeat definition of the function
  This is the end of the script

>>>
```

Retrieving data from functions

- Python uses the return statement to exit a function with a specific value.

functions4.py

```
def double_value():
    value = int (input("Enter a value: "))
    print ("Doubling the value")
    result = value * 2
    return result

x = double_value()
print ("The new value is: ", x)
```

Shell

```
>>> %Run functions4.py

  Enter a value: 5
  Doubling the value
  The new value is:  10

>>> |
```

Passing data to functions

- You need to pass values into a function from your main program by using arguments.
- Arguments are values enclosed within the function parentheses, like this:

```
result = addtwo (3, 20)
```

Passing data to functions

- To retrieve the argument values in your Python functions, you define parameters in the function definition.
- Parameters are variables you place in the function definition to receive the arguments values when the main program calls the function

```
def addtwo (a, b):
    result = a + b
    return result
```

Passing data to functions

functions5.py

```
def addtwo (a, b):
    result = a + b
    return result

total = addtwo (5, 10)
print ("The total is: ", total)
```

Shell

```
>>> %Run functions5.py
  The total is:  15
>>>
```

Passing data to functions

- If you don't provide any arguments, or if you provide the incorrect number of arguments, you get an error message from Python.

functions5.py

```python
def addtwo (a, b):
    result = a + b
    return result

total = addtwo (5, 10)
print ("The total is: ", total)

total1 =  addtwo()

total2 = addtwo (10, 20, 30)
```

Shell

```
>>> %Run functions5.py
  The total is:  15
  Traceback (most recent call last):
    File "/home/pi/Documents/scripts/functions5.py", line 8, in <module>
      total1 =  addtwo()
  TypeError: addtwo() missing 2 required positional arguments: 'a' and 'b'
>>>
```

Passing data to functions

- Default parameters
  - Python allows you to set default values assigned to parameters if no arguments are provided when the main program calls the function.
  - You must set the default values inside the function definition.

funtions6.py

```
def area (width = 10, height = 10):
    area = width * height
    return area

total = area (15, 15)
print ("The area is: ", total)

total1 = area (15)
print ("The area is: ", total1)

total2 = area ()
print ("The area is: ", total2)
```

Shell

```
>>> %Run funtions6.py

  The area is:  225
  The area is:  150
  The area is:  100

>>>
```

Passing data to functions

- Dealing with a Variable Number of Arguments
  - In some situations, you might not have a set number of parameters for a function.
  - Instead, a function might require a variable number of parameters.
  - You can accommodate this by using the following special format to define the parameters:

```
def funct (*args)
```

perimeter.py

```python
def perimeter (*args):
    sides = len (args)
    print ("There are: ", sides, "sides to the object.")
    total = 0
    for i in range (0, sides):
        total = total + args[i]
    return total

triangle = perimeter (4,6,10)
print ("The perimeter of the triangle is: ", triangle)

rectangle = perimeter (4,8,4,8)
print ("The perimeter of the rectangle is: ", rectangle)

octagon = perimeter (4,6,2,3,4,6,2,3)
print ("The perimeter of the octagon is: ", octagon)
```

Shell

```
>>> %Run perimeter.py
 There are:  3 sides to the object.
 The perimeter of the triangle is:  20
 There are:  4 sides to the object.
 The perimeter of the rectangle is:  24
 There are:  8 sides to the object.
 The perimeter of the octagon is:  30

>>>
```

Passing data to functions

- Retrieving Values Using Dictionaries
  - You can use a dictionary variable to retrieve the argument values passed to a function.
  - To do this, you place two asterisks (**) before the dictionary variable name in the function parameter:
  - `def function (**kwargs)`

volume.py

```python
def volume (**kwargs):
    radius = kwargs['radius']
    height = kwargs['height']
    print ("The radius is: ", radius)
    print ("The height is: ", height)
    total = 3.141592 * radius * radius * height
    return total

cylinder = volume (radius=5, height=30)
print ("The volume of the cylinder is: ", cylinder)
```

Shell

```
>>> %Run volume.py

  The radius is:  5
  The height is:  30
  The volume of the cylinder is:  2356.194

>>>
```

Handling Variables in a Function

- As you start writing more complex Python scripts, you'll find yourself reusing parts of code that perform specific tasks.
- Python provides a feature that lets you reuse the block of code.
- Any time you need to use that block of code in your script, you simple use the name you assigned to the function.
- This is referred to as calling the function.

Handling Variables in a Function

- There are two types of variables in Python:
  - Local Variables
    - Variables created inside a function. You can access only inside the function
  - Global Variables
    - Variables you can use anywhere in your program code, including inside the functions.

Handling Variables in a Function

global.py

```
width = 10
height = 60
total = 0

def area():
    total = width * height
    print ("Inside the function the total is: ", total)

area()
print ("Outside the function the total is: ", total)
```

Shell

```
>>> %Run global.py

  Inside the function the total is:  600
  Outside the function the total is:  0

>>>
```

Handling Variables in a Function

- When you try to read the total variable in the main program, the value is set back to the global value assignment…
- To solve this problem, you need to say to Python that the function is trying to access a global variable.

Handling Variables in a Function

global.py

```python
width = 10
height = 60
total = 0

def area():
    global total
    total = width * height
    print ("Inside the function the total is: ", total)

area()
print ("Outside the function the total is: ", total)
```

Shell

```
>>> %Run global.py

  Inside the function the total is:  600
  Outside the function the total is:  600

>>>
```

Handling Variables in a Function

- Using List with Functions
  - When you pass values as arguments to functions, Python passes the actual value, not the variable location in memory; this is called passing by reference.
  - If you pass a mutable object (such as a list or dictionary variable), the function can make changes to the object itself.

Handling Variables in a Function

- Using List with Functions

functionList.py *

```
def modlist(x):
    x.append('Jason')

mylist = ['Richard','Christie']
print ("The list before the function call: ", mylist)
modlist(mylist)
print ("The list after the function call: ", mylist)

```

Shell

```
>>> %Run functionList.py

  The list before the function call:  ['Richard', 'Christie']
  The list after the function call:  ['Richard', 'Christie', 'Jason']

>>>
```

Using Recursion with Functions

- A popular use of functions is in a process called recursion.
- In recursion, you solve an algorithm by repeatedly breaking the algorithm into subsets until you reach a core definition value.

factorial.py

```
def factorial (num):
    if (num == 0):
        return 1
    else:
        return num * factorial (num - 1)

result = factorial (5)
print ("The factorial of 5 is: ", result)
```

Shell

```
>>> %Run factorial.py
 The factorial of 5 is:  120
>>>
```

Exercises

Lab 5

- Goals:
  - How to use the for loop
  - How to use the while loop
  - How to use nested loops
  - Creating your own functions
  - Retrieving data from functions
  - Passing data to functions
  - Using lists with functions
  - Using functions in your Python Scripts

Lab 4

- 1. Answer the followings questions:
  - A for loop is a count-controlled loop, and a while loop is a condition-controlled loop. Explain the differences.
  - What is wrong with the following code's syntax?

```
for a_number in [5, 10, 1]
    print (a_number)
```

  - What's happened when a break command is executed in a while loop?

Lab 4

- 2. Answer the followings questions:
  - You must define a function before you can use it in your Python script. True or False? Explain your answer.
  - How do you define a default value for a function parameter.
  - How do you define a variable number of parameters in a function definition?
  - Local variables can only be referenced from inside the function where they are created. True or False. Explain your answer.

Lab 4

- 3. Write a script that gives by default the approximate value of the mathematical constant e, for a n value, using the formula:

$$e \approx \sum_{i=0}^{n} \frac{1}{i!}$$

- To do this:
  - Define the factorial(n) function
  - Define the value_e(n) function
  - In your main program, prompt the user for the n value, call the value_e (n) function
  - Display the corresponding approximation of e at the screen.

Lab 4

- 4. Create a function named pyramid (n) that draws a pyramid like this:

```
*
***
*****
*******
*********
```

where n is the number of lines to draw. In the case shown n = 5.

Lab 4

- 5. Modify your function pyramid to be able to receive optionally a new parameter c. You will create the pyramid with the char c:

```
x
xxx
xxxxx
xxxxxxx
xxxxxxxxx
```

where n is the number of lines to draw. In the case shown n = 5 and c= x

I should be able to use the old method with only n as parameter.

Lab 4

- 6. Create a function is_prime (n) that takes as argument a positive integer (greater than 2) and that returns the Trues if the value n is prime and False if is not prime.

- Determine all the prime numbers from 2 to 100. We want to have an output similar to this:

```
2 is prime
3 is prime
4 is not prime
[...]
100 is not prime
```

Lab 4

- 7. Write the function solve_cuadratic (a,b,c) that calculates roots of the second-degree equation of the form: $ax^2 + bx + c$.

The function receives the three parameters of the trinomial, a, b and c.

The function must return a tuple whose first element is the number of roots of the trinomial (0, 1 or 2), and the other elements are the possible roots.

Test your function with the following three sets of values: 1, −3, 2, 1, −2, 1 and 1, 1, 1.

Are you able to use the complex numbers in this function?